What is claimed is:

1. A method comprising:

generating an intermediate representation (IR) of a source program, where the source program includes one or more instructions for processing data in a bit field within a data structure;

modifying the intermediate representation to more efficiently execute the one or more instructions for processing the bit field data; and

generating resultant code based on the modified intermediate representation.

2. The method of claim 1, wherein modifying the intermediate representation further comprises:

pre-processing the IR to perform preliminary modification of the IR.

3. The method of claim 2, wherein modifying performing pre-processing further comprises:

performing data flow analysis to gather information regarding definition and usage of the bit field data; and

generating a def/use graph to classify the information.

4. The method of claim 3, wherein generating a def/use graph further comprises:

generating a def/use graph to classify the information in relation to an associated packet.

5. The method of claim 2, wherein modifying the intermediate representation further comprises:

(a) allocating a temporary variable to hold the bit field data; and

(b) modifying the IR so that the temporary variable is processed in accordance with the instructions.

6. The method of claim 5, further comprising:

(c) assigning the value of the temporary variable to a memory.

7. The method of claim 6, further comprising:

performing steps (a), (b) and (c) for a single basic block.

8. The method of claim 7, further comprising:

identifying two or more sub-blocks within the basic block.

9. The method of claim 8, wherein:

steps (a) , (b) and (c) are performed for each sub-block.

10. The method of claim 5, further comprising:

determining whether all of the one or more instructions for processing the bit field data are read-after-write instructions; and

performing steps (a) and (b) only if the determination is false.

11. The method of claim 6, further comprising:

determining whether any of the one or more instructions for processing the bit field data are write instructions; and

performing step (c) only if the determination is true.

12. The method of claim 6, further comprising:

removing the modifications effected by steps (a), (b) and (c) upon determining that such removal is expected to provide an efficiency benefit in the resultant code.

13. The method of claim 2, wherein pre-processing further comprises:

disambiguating a memory reference to the bit field.

14. The method of claim 1, wherein modifying the intermediate representation further comprises:

modifying the IR so that multiple instructions to initialize respective bit fields of a data structure are performed with a single write to a memory.

15. The method of claim 14, wherein the multiple instructions occur within a pre-defined maximal scope.

16. The method of claim 1, wherein modifying the intermediate representation further comprises:

modifying the IR so that multiple read instructions for respective bit fields of a data structure are performed with a single read from a memory.

17. The method of claim 16, wherein the multiple read instructions occur within a pre-defined maximal scope.

18. The method of claim 1, wherein modifying the intermediate representation further comprises:

modifying the IR so that multiple write instructions to respective bit fields of a data structure are performed with a single write to a memory.

19. The method of claim 18, wherein the multiple read instructions occur within a pre-defined maximal scope.

20. The method of claim 1, wherein modifying the intermediate representation further comprises:

determining that a first instruction, being one of the one or more instructions, indicates a bit-wise logical operation on the bit field data;

determining that a second instruction of the source program indicates a bit-wise logical operation on a second bit field within the data structure; and

modifying the IR so that the first and second instructions are performed via a single read from a memory.

21. The method of claim 20, wherein the bit-wise logical operation is a bit-wise OR operation.

22. The method of claim 20, wherein the bit-wise logical operation is a bit-wise AND operation.

23. An article comprising:

a machine-readable storage medium having a plurality of machine accessible instructions, which if executed by a machine, cause the machine to perform operations comprising:

generating an intermediate representation (IR) of a source program, where the source program includes one or more instructions for processing data in a bit field within a data structure;

modifying the intermediate representation to more efficiently execute the one or more instructions for processing the bit field data; and

generating resultant code based on the modified intermediate representation.

24. The article of claim 23, wherein the instructions that cause the machine to modify the intermediate representation further comprise instructions that cause the machine to:

perform preliminary modification of the IR.

25. The article of claim 24, wherein the instructions that cause the machine to modify the intermediate representation further comprise instructions that cause the machine to:

gather information regarding definition and use of the bit field data; and

generate a def/use graph to classify the information.

26. The article of claim 25, wherein the instructions that cause the machine to generate a def/use graph further comprise instructions that cause the machine to:

generate a def/use graph to classify the information in relation to an associated packet.

27. The article of claim 24, wherein the instructions that cause the machine to modify the intermediate representation further comprise instructions that cause the machine to:

(a) allocating a temporary variable to hold the bit field data; and

(b) modifying the IR so that the temporary variable is processed in accordance with the instructions.

28. The article of claim 27, further comprising a plurality of machine accessible instructions, which if executed by a machine, cause the machine to perform operations comprising:

(c) assigning the value of the temporary variable to a memory.

29. The article of claim 28, further comprising a plurality of machine accessible instructions, which if executed by a machine, cause the machine to perform operations comprising:

performing steps (a), (b) and (c) for a single basic block.

30. The article of claim 29, further comprising a plurality of machine accessible instructions, which if executed by a machine, cause the machine to perform operations comprising:

identifying two or more sub-blocks within the basic block.

31. The article of claim 30, further comprising a plurality of machine accessible instructions, which if executed by a machine, cause the machine to perform operations comprising:

performing steps (a) , (b) and (c) for each sub-block.

32. The article of claim 27, further comprising a plurality of machine accessible instructions, which if executed by a machine, cause the machine to perform operations comprising:

determining whether all of the one or more instructions for processing the bit field data are read-after-write instructions; and

performing steps (a) and (b) only if the determination is false.

33. The article of claim 28, further comprising a plurality of machine accessible instructions, which if executed by a machine, cause the machine to perform operations comprising:

determining whether any of the one or more instructions for processing the bit field data are write instructions; and

performing step (c) only if the determination is true.

34. The article of claim 28, further comprising a plurality of machine accessible instructions, which if executed by a machine, cause the machine to perform operations comprising:

removing the modifications effected by steps (a), (b) and (c) upon determining that such removal is expected to provide an efficiency benefit in the resultant code.

35. The article of claim 24, wherein the instructions that cause the machine to perform preliminary modification of the IR further comprise instructions that cause the machine to:

disambiguate a memory reference to the bit field.

36. The article of claim 23, wherein the instructions that cause the machine to modify the intermediate representation further comprise instructions that cause the machine to:

modify the IR so that multiple instructions to initialize respective bit fields of a data structure are performed with a single write to a memory.

37. The article of claim 36, wherein the multiple instructions occur within a pre-defined maximal scope.

38. The article of claim 23, wherein the instructions that cause the machine to modify the intermediate representation further comprise instructions that cause the machine to:

modify the IR so that multiple read instructions for respective bit fields of a data structure are performed with a single read from a memory.

39. The article of claim 38, wherein the multiple read instructions occur within a pre-defined maximal scope.

40. The article of claim 23, wherein the instructions that cause the machine to modify the intermediate representation further comprise instructions that cause the machine to:

modify the IR so that multiple write instructions to respective bit fields of a data structure are performed with a single write to a memory.

41. The article of claim 40, wherein the multiple read instructions occur within a pre-defined maximal scope.

42. The article of claim 23, wherein the instructions that cause the machine to modify the intermediate representation further comprise instructions that cause the machine to:

determine that a first instruction, being one of the one or more instructions, indicates a bit-wise logical operation on the bit field data;

determine that a second instruction of the source program indicates a bit-wise logical operation on a second bit field within the data structure; and

modify the IR so that the first and second instructions are performed via a single read from a memory.

43. The article of claim 42, wherein the bit-wise logical operation is a bit-wise OR operation.

44. The article of claim 42, wherein the bit-wise logical operation is a bit-wise AND operation.

45. A compiler comprising:

a front end to generate an intermediate representation of a source program;

an optimizer to modify the intermediate representation (IR) to provide for optimized processing of one or more bit fields; and

a back end to generate resultant code based on the modified intermediate representation.

46. The compiler of claim 45, wherein:

the optimizer includes a pre-processor to perform preliminary processing of the intermediate representation.

47. The compiler of claim 46, wherein:

the pre-processor includes a data flow analyzer to perform data flow analysis and to generate a def/use graph.

48. The compiler of claim 46, wherein:

the pre-processor includes a registerizer to modify the intermediate representation to allocate a temporary variable for a bit field variable used in the source program.

49. The compiler of claim 47, further comprising:

an unregisterizer to selectively reverse the modification performed by the registerizer.

50. The compiler of claim 45, wherein:

the optimizer includes a bit-specific optimizer to modify the IR such that processing of bit fields indicated by the source program is more efficient.

51. The compiler of claim 50, wherein:

the bit-specific optimizer includes an aggregate initializer to initialize multiple bit fields within a data structure via a single write to memory.

52. The compiler of claim 50, wherein:

the bit-specific optimizer includes a read/write combiner to read multiple bit fields within a data structure via a single read from memory.

53. The compiler of claim 52, wherein:

the read/write combiner is further to initialize write bit fields within a data structure via a single write to memory.

54. The compiler of claim 50, wherein:

the bit-specific optimizer includes a juxtaposition merger to determine that a first instruction, being one of the one or more instructions, indicates a bit-wise logical operation on the bit field data;

the juxtaposition merger further to determine that a second instruction of the source program indicates a bit-wise logical operation on a second bit field within the data structure; and

the juxtaposition optimizer further to modify the IR so that the first and second instructions are performed via a single read from a memory.

55. The compiler of claim 50, wherein:

the bit-specific optimizer includes an "or" optimizer to merge logical "or" statements of a conditional statement together such that they are executed via a single read statement.

56. The compiler of claim 55, wherein:

the "or" optimizer is further to merge bit-wise "or" statements of a conditional statement together such that they are executed via a single read statement.

57. The compiler of claim 50, wherein:

the bit-specific optimizer includes an "and" optimizer to merge logical "and" statements of a conditional statement together such that they are executed via a single read statement